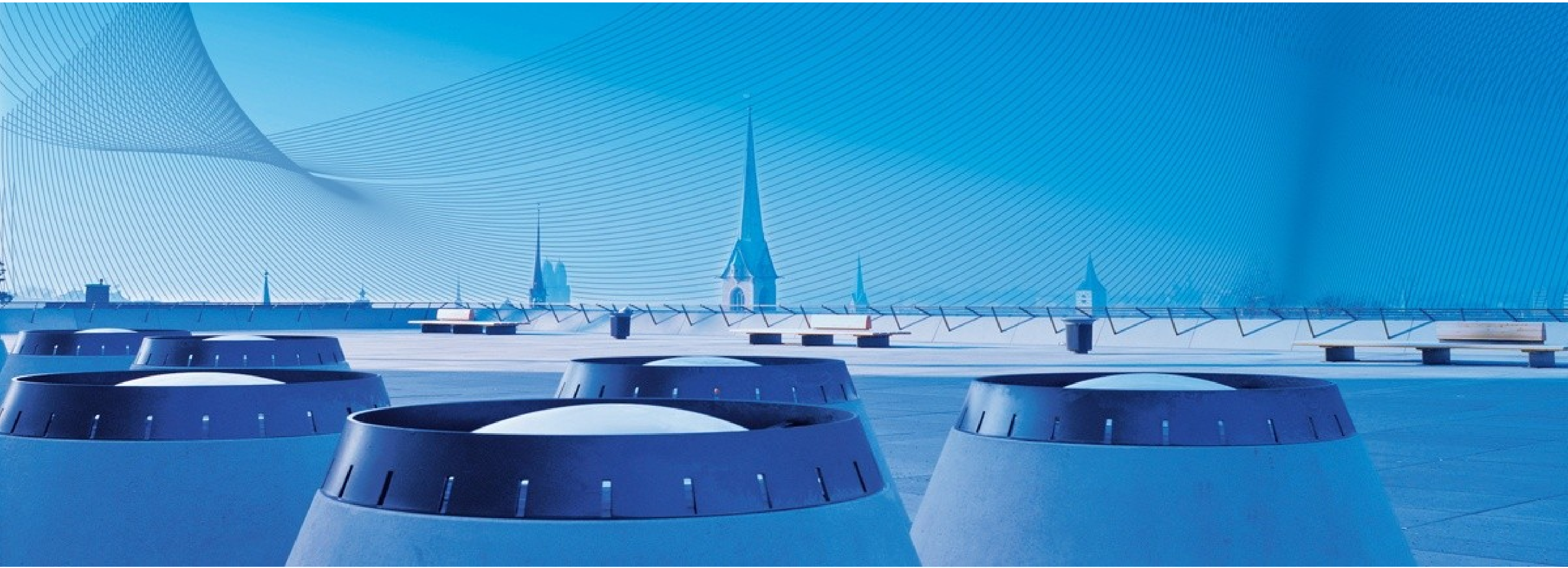
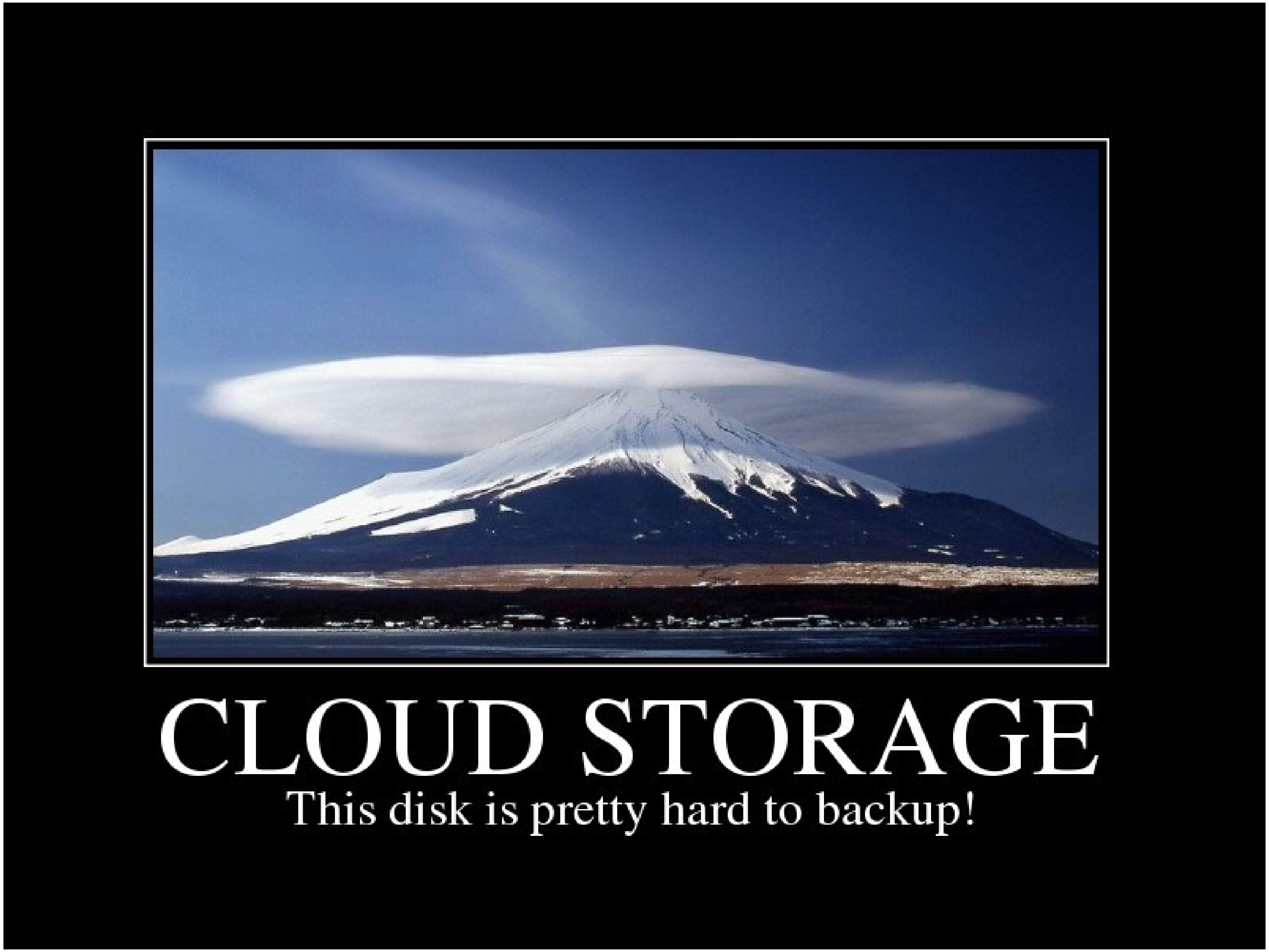
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Databases

Seminar in Distributed Computing 08
with papers chosen by Prof. D. Kossmann

Nico Waldispühl





- Goal: Overview over current state of ideas in cloud storage by showing some selected aspects of three examples of distributed systems

Content overview

- Introduction / Motivation
- How Amazon implemented a simple distributed database service
- Relational database on top of simple distributed database service
- How Google implemented a locking service
- Conclusion, References

Introduction

Conventional business (i.e. selling goods) bases on physical objects:

- Mostly regional (if not, significant delay for delivery) (transaction time: days)
- Handling restricted by physical laws: Only (small) finite number of people in your shop at the same time, only finite number of objects in stock.
- Slow (= manageable) reactions on success/failure (weeks)

> Plenty of time to react on a trend after noticing it!

Introduction

E-business (providing services) bases on virtual objects (i.e. information):

- Available world-wide, technically the whole earth population as potential customers (transaction time: seconds)
- Success can come very fast (hours) i.e. by reviews in online media generating a hype.

> Practically no time to react properly.

Thus: **Success can kill!**

Introduction

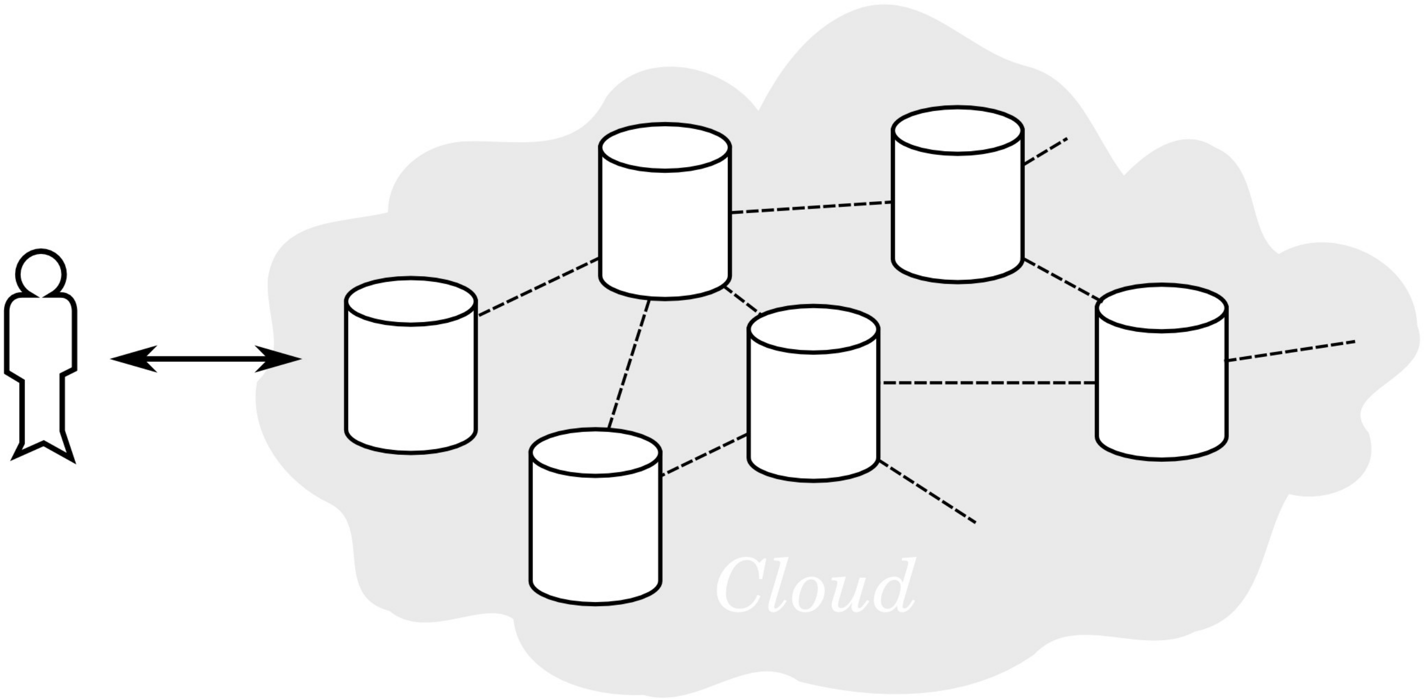
How to be prepared for a possible success of your e-business?

- Try to anticipate the turnaround?
  > Not reliable.
- Buy server infrastructure in advance?
  > May be misinvestment if your idea doesn't pay.
- Just wait until the success comes and invest after?
  > If you're offline for more than some hours, your reputation is lost.

No way to manage instant success? Fortunately yes...

Cloud Computing (Utility Computing)

- 'Outsourcing' computation, storage and network to a service provider which leases them to the customer.



Cloud Computing (Utility Computing)

- Service provider maintains data centers all over the world-> Position of data ist never exactly clear-> Cloud
- For external observers: „Intelligence“ goes from the border back into the net.

Cloud Computing (Utility Computing)

Huge benefits. Such distributed services usually have these properties:

- Unlimited scalability (~)
  - Datacenters designed for very large traffic
  - Load balancing
- Always available
  - Heavily distributed -> fail safe
- You only pay what you need/use
  - Billing by consumed space/processortime.
  - No need to operate own hardware

(Also various risks and downsides to consider; i.e. privacy, loss of control)

Cloud Computing (Utility Computing)

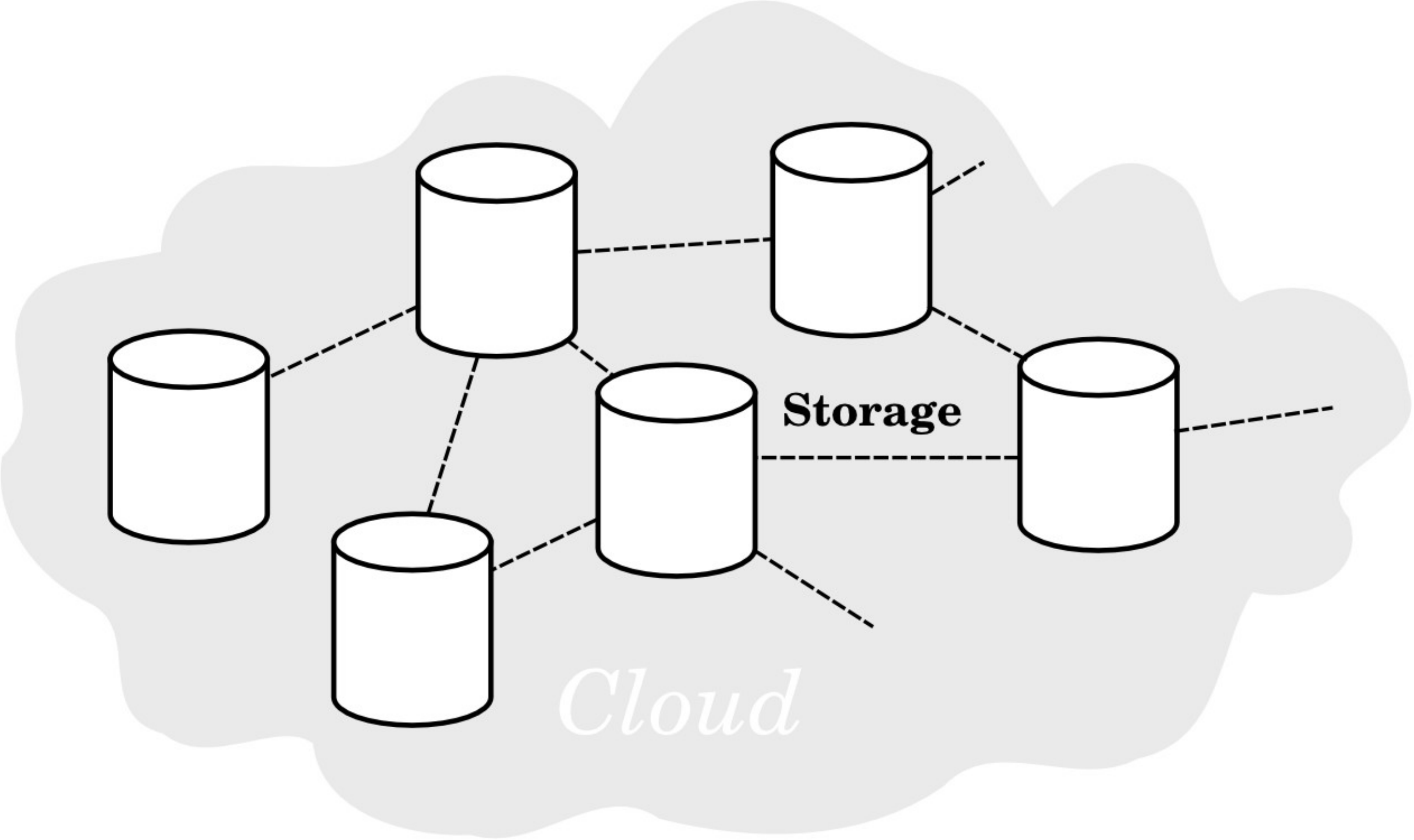
In theory already known some time. In practice evolved as by-product of the dot-com bubble:

- Amazon (among others) heavily upgraded their data centers around 2001/02
- New architectures lead to overcapacities.
- Parts of the infrastructure now leasable under the term AWS – Amazon Web Services:
  - EC2 – Elastic Compute Cloud
  - S3 – Simple Storage Service
  - SQS – Simple Query Service

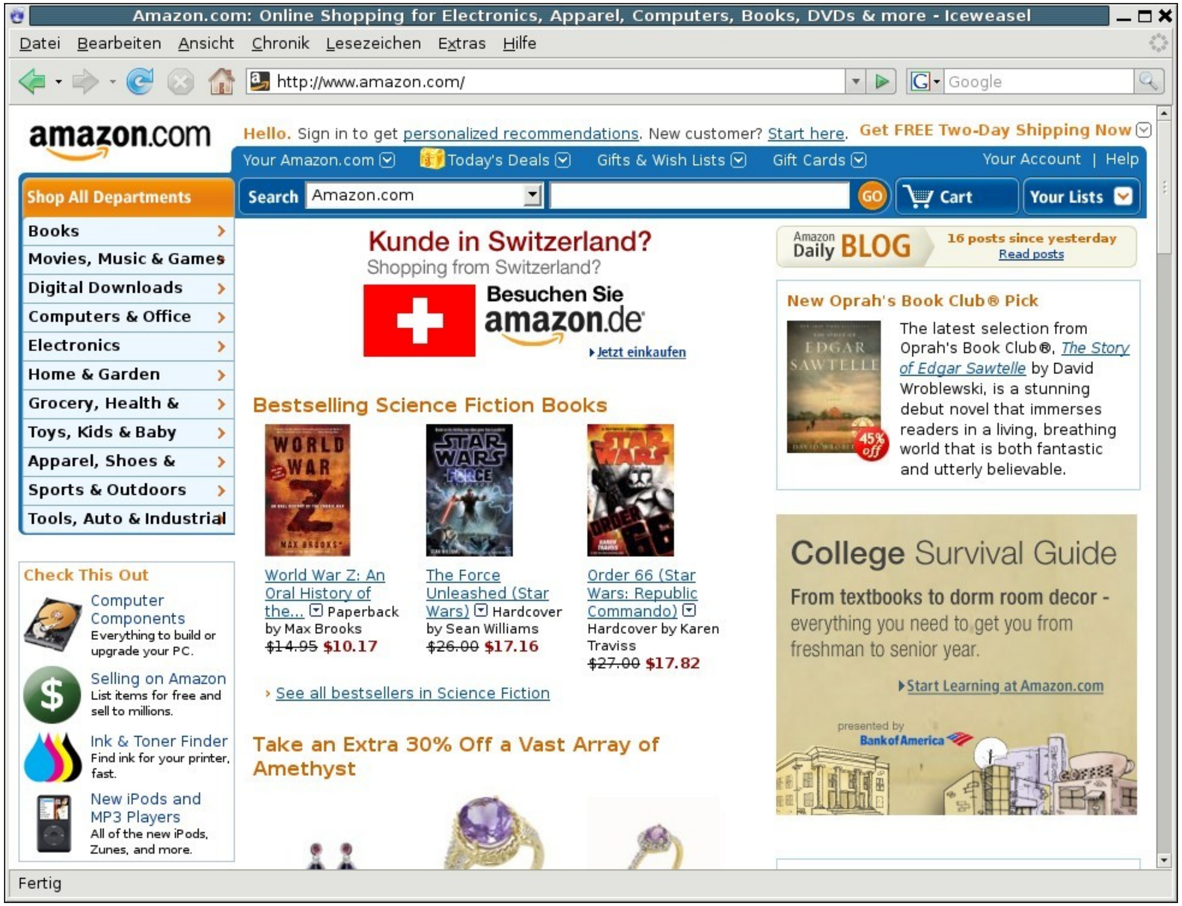
amazon.com

powered by amazon web services™

How is such cloud storage implemented?



Dynamo: Amazon's internal solution



Hello. Sign in to get personalized recommendations. New customer? Start here.

Get FREE Two-Day Shipping Now

Kunde in Switzerland?

Shopping from Switzerland?

Besuchen Sie amazon.de

Jetzt einkaufen

Amazon Daily BLOG

16 posts since yesterday

Read posts

New Oprah's Book Club® Pick

The latest selection from Oprah's Book Club®, *The Story of Edgar Sawtelle* by David Wroblewski, is a stunning debut novel that immerses readers in a living, breathing world that is both fantastic and utterly believable.

Bestselling Science Fiction Books

World War Z: An Oral History of the... Paperback by Max Brooks ~~$14.95~~ $10.17

The Force Unleashed (Star Wars) Hardcover by Sean Williams ~~$26.00~~ $17.16

Order 66 (Star Wars: Republic Commando) Hardcover by Karen Traviss ~~$27.00~~ $17.82

See all bestsellers in Science Fiction

Check This Out

Computer Components
Everything to build or upgrade your PC.

Selling on Amazon
List items for free and sell to millions.

Ink & Toner Finder
Find ink for your printer, fast.

New iPods and MP3 Players
All of the new iPods, Zunes, and more.

Take an Extra 30% Off a Vast Array of Amethyst

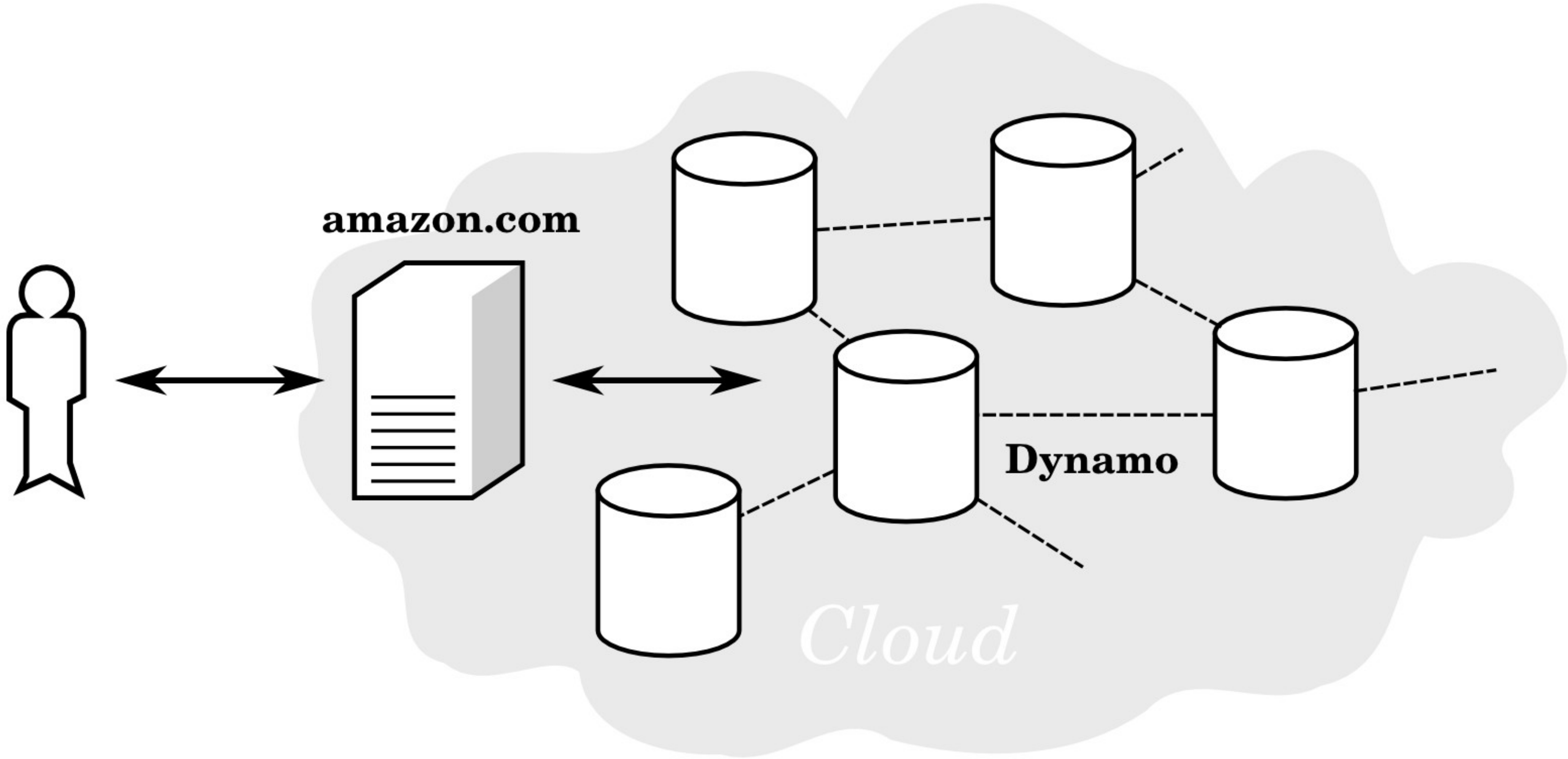
College Survival Guide

From textbooks to dorm room decor - everything you need to get you from freshman to senior year.

Start Learning at Amazon.com

presented by Bank of America

What is Dynamo?



- Amazons highly available distributed key-value store

Why key-value?

The low complexity of a key-value store leads to:

- Increased speed (No query engine)
- Better scalability (Load balancing is done easier)
- Better Maintainability

> Since many of Amazons services only save data by primary key, more complex systems would be waste of resources.

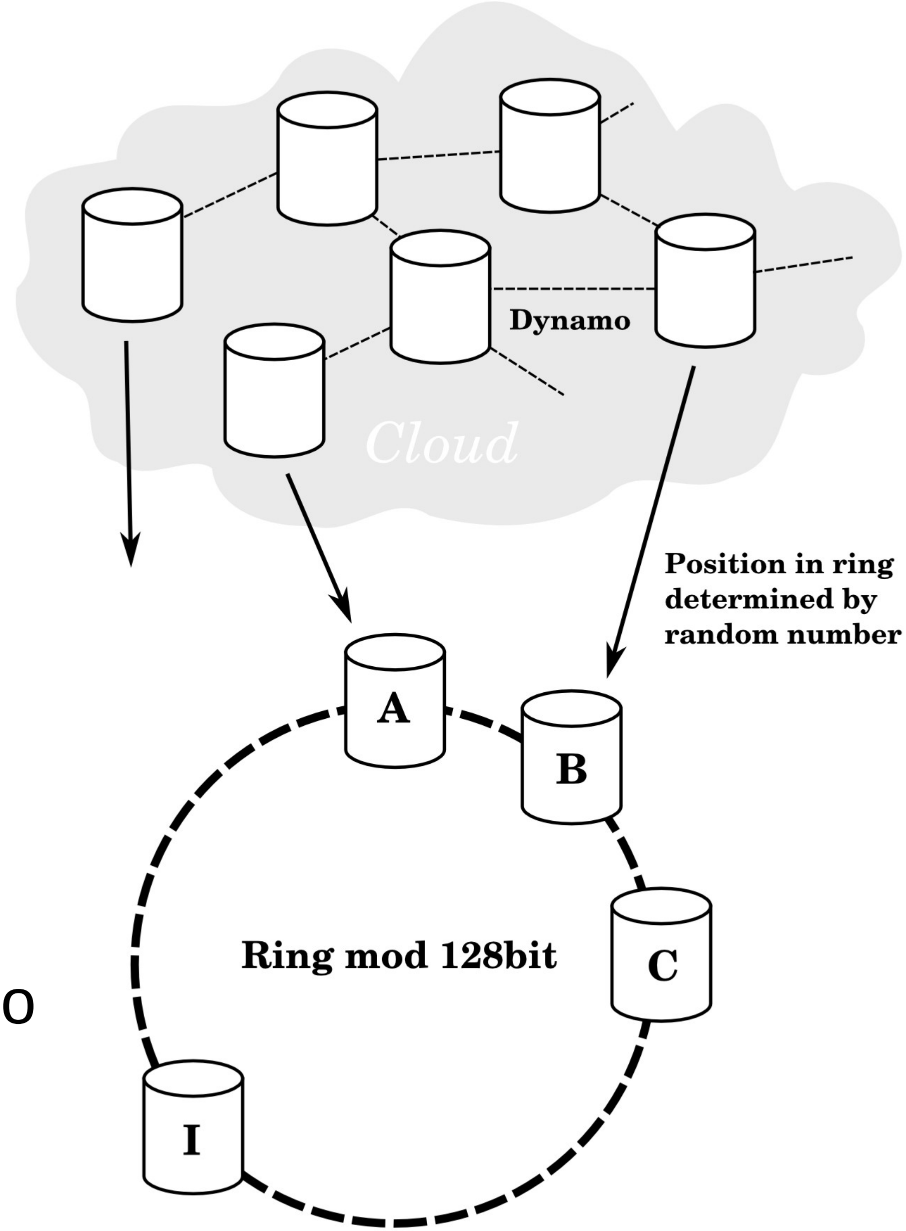
Dynamo characteristics

- High availability, reliability & performance
- Eventual consistency

- Applications using dynamo can trade off availability, consistency, cost-effectiveness and performance by choosing some system parameters on their own.

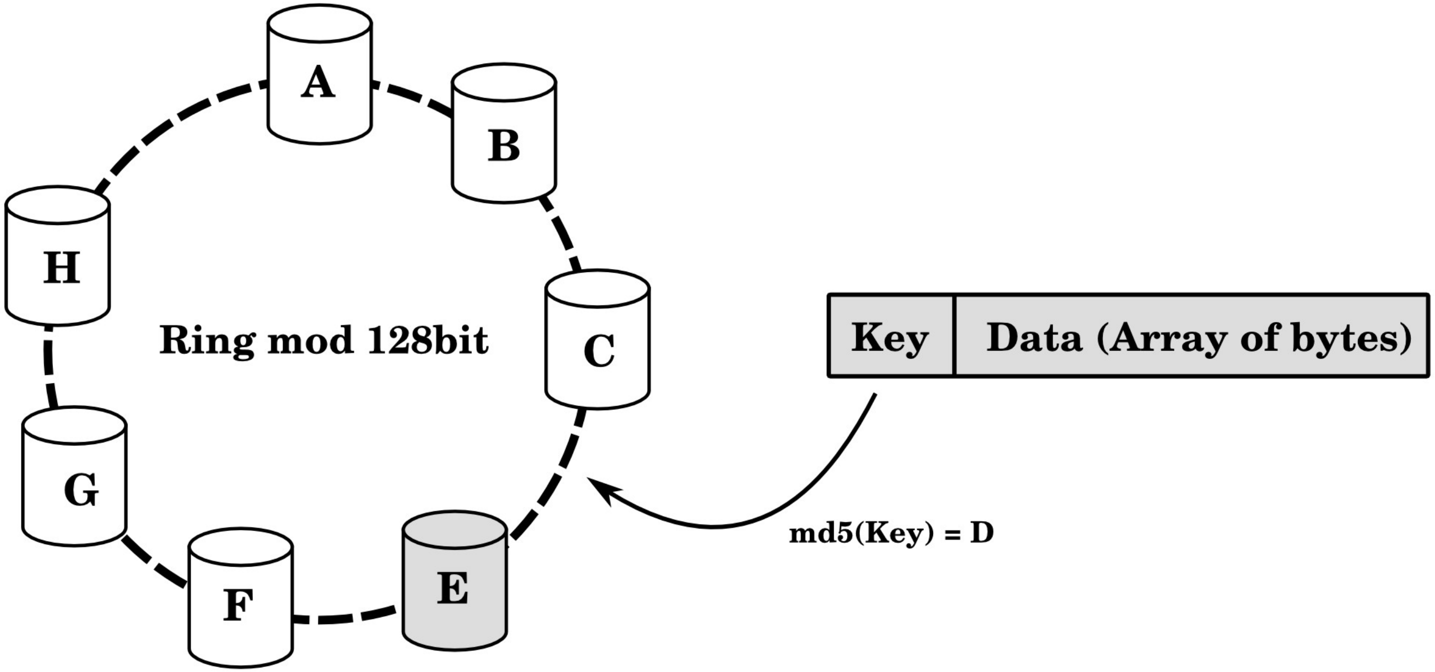
What is dynamos architecture like?

- Nodes choose random number
- According to this number they are positioned in a ring
- (The reality is a bit more complex: Each physical node is divided to multiple virtual nodes in the ring)



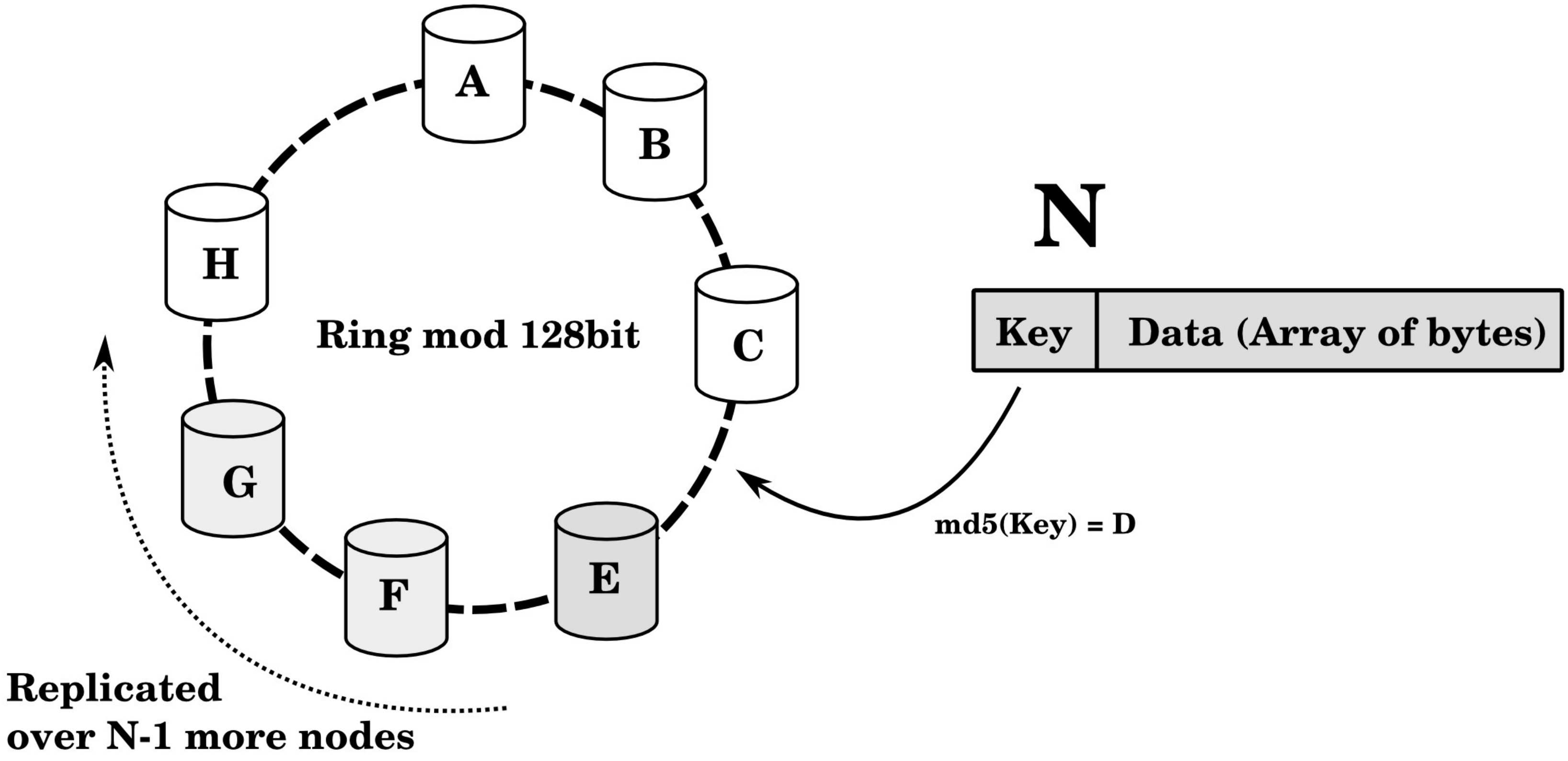
How is data stored?

- A data items consists of a key and the data payload.
- The key is hashed -> 128 bit identifier.
- First node with *position* >= *key hash* is responsible



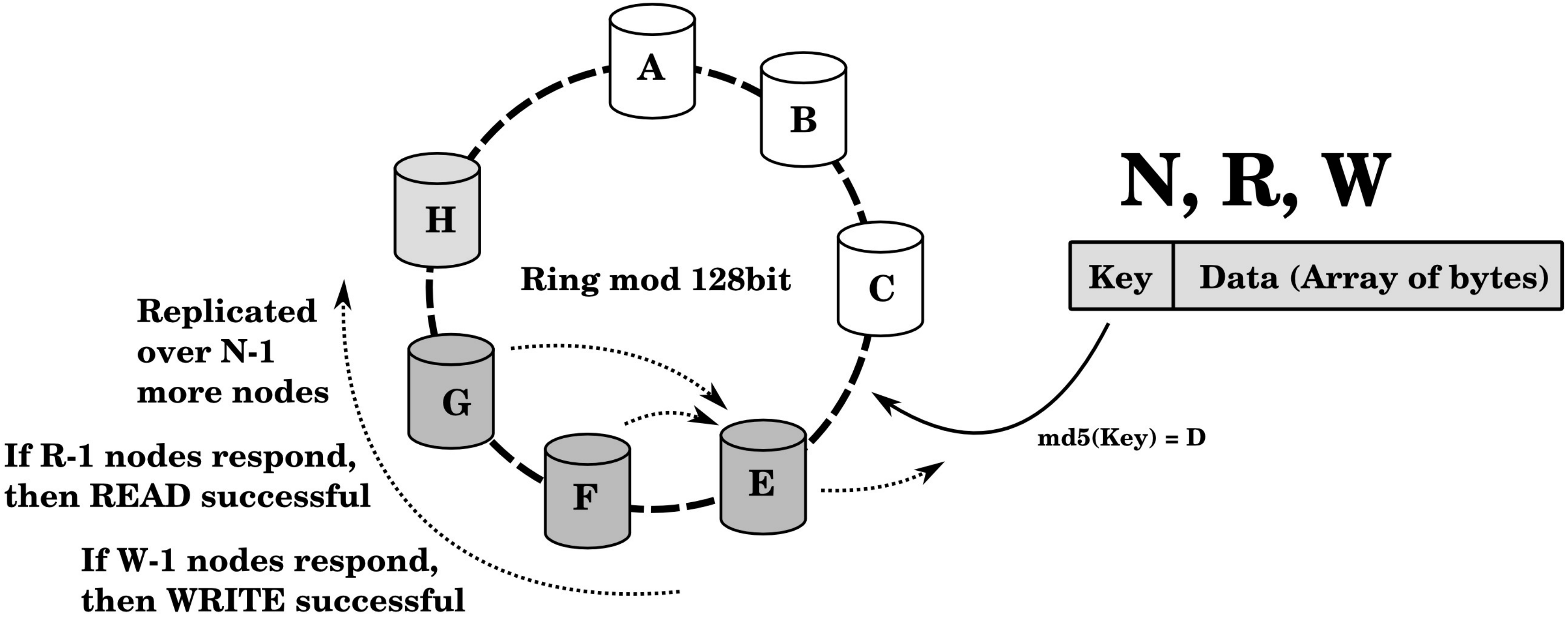
How is the data replicated?

- Applications can choose a value N
- Data is stored on first N healthy nodes



How maintain consistency among nodes?

- Dynamo uses a sloppy quorum system with two parameters R and W
- R and W state the minimum of nodes to participate on a successful *R*ead or *W*rite operation



How to deal with different data versions?

- Can be affected by choosing N,R,W accordingly
- Updates are replicated asynchronously
- Network partitions or node failures can lead to several versions of the same data

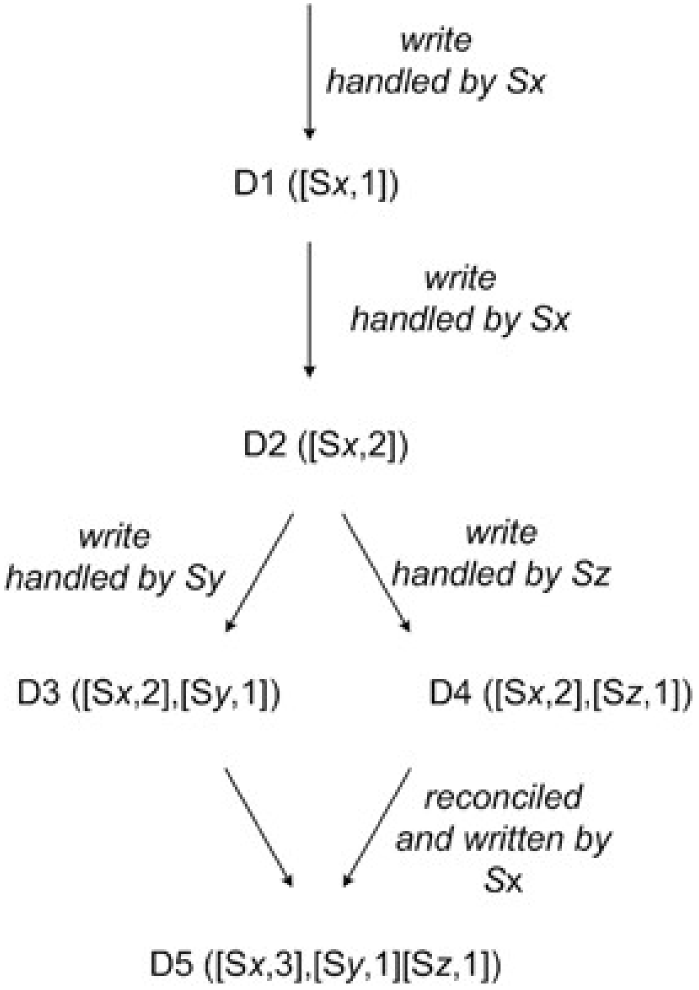
->Dynamo uses *vector clocks* to reconcile multiple versions of data. -> On each update of a data item, a vector clock timestamp is added.

Vector clock timestamp: **Version(**List of **NV-Pairs)**

Node/Version-Pair: **[Writing Node, Item Version]**

How do these vector clocks work?

- Every write adds a version tuple (a context)
- If there are:
  - Two or more concurrent writes from the same node, the highest is taken
  - Two or more concurrent writes from different nodes, all of them are returned
- A dataset is considered reconciled if a node updates such context



Does Dynamo perform well enough?

- According to Amazon people:
  - No data loss event has ever occurred
  - 99.9995% of requests were successful (no time-out)
  - Great adaptability with choosable parameters (N, R, W)
  - Currently (2007) a couple of hundreds of nodes run without greater problems. But: Tens of thousands of nodes problematic because of the routing tables (hash mappings)

    > To be introduced: Hierarchical extensions

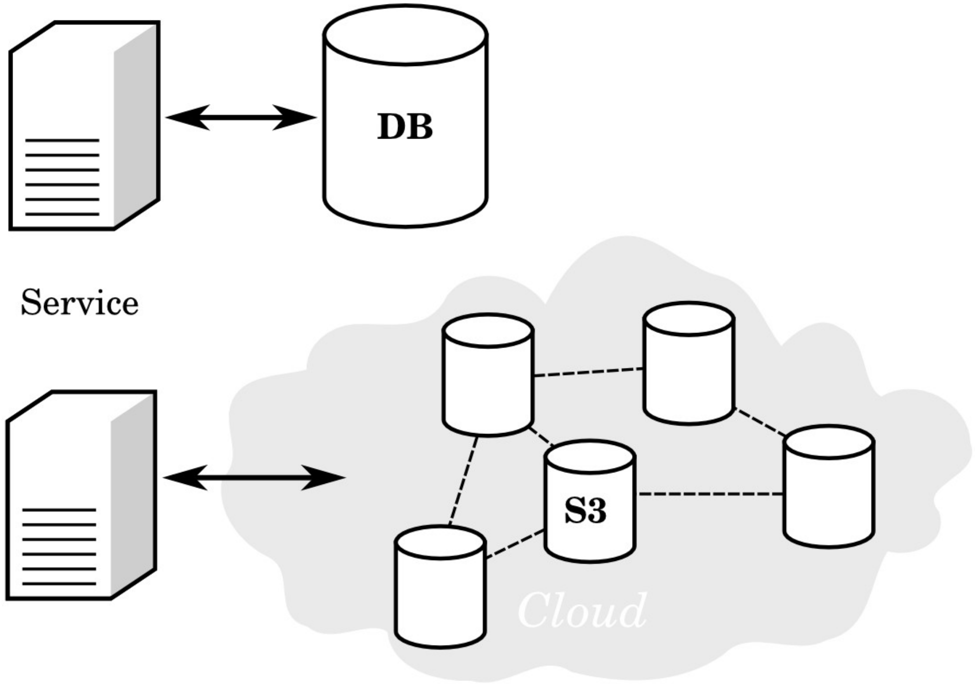
S3: Amazons external solution

- People may want to use such store for their businesses
- A service similar to Dynamo is available for customers: S3
- Except for the parameters N,R,W pretty much the same specification
- How can it be used for applications / web services that rely on relational database schemes?

Building a Database on S3

Using S3 as backend for a database for web services:

- How is it implemented?
- Can it be made reliable?
- Does it pay?



What exactly is S3?

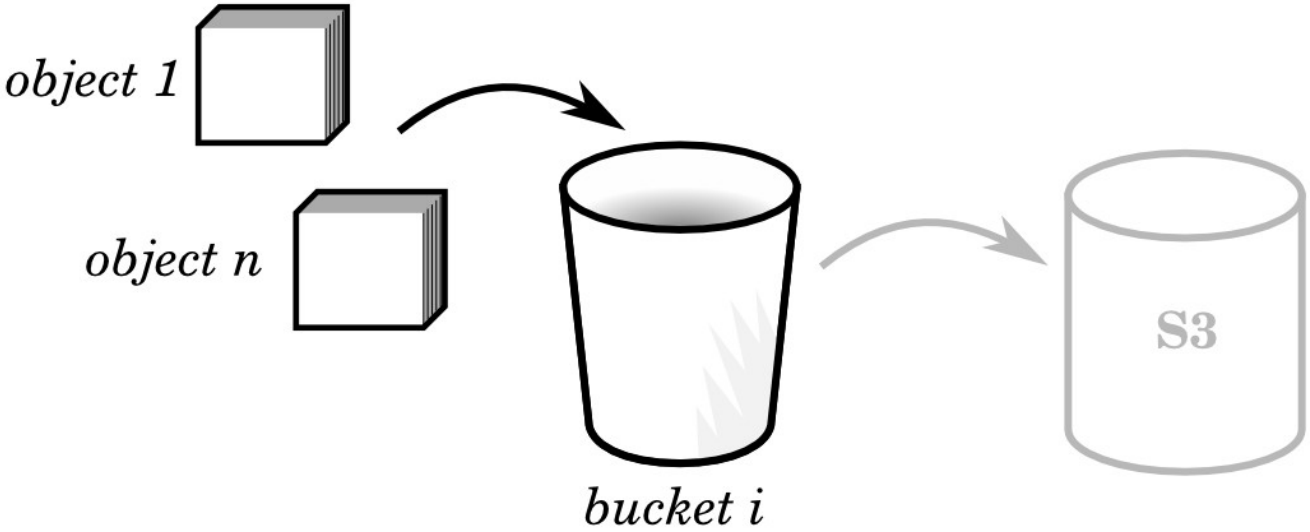
S3 is Amazons distributed key-value database service.

- Infinite store for single objects (size: [1, 5G] byte)
- Unlimited availability (No request is ever blocked!)
- Unlimited scalability

But:

- Only eventual consistency guaranteed!
- Cost per storage/time, transfer and # of transactions

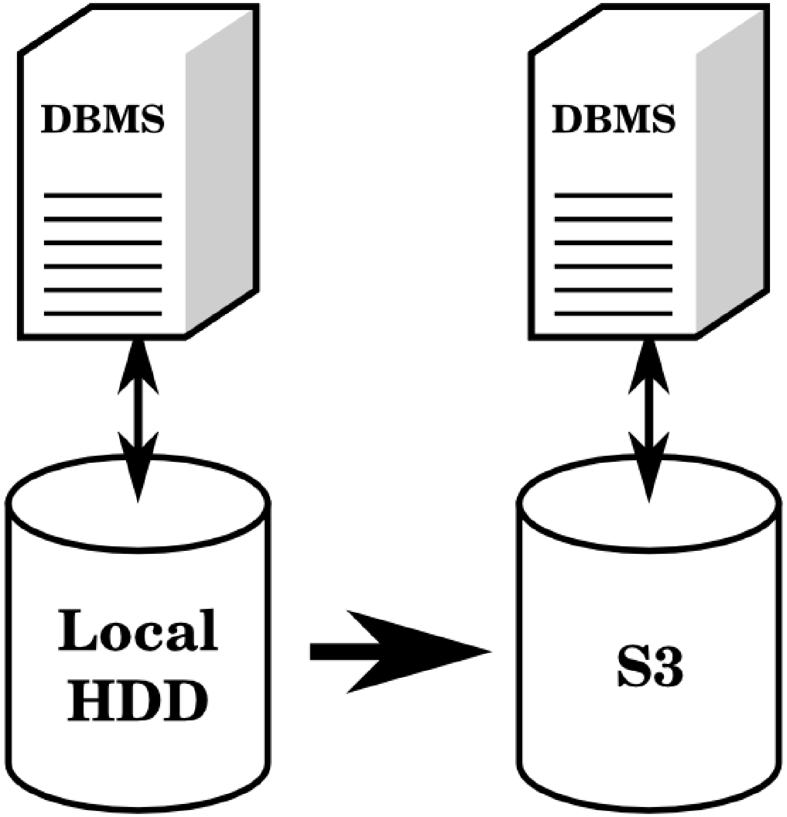
How does S3 work conceptually?



- Objects in buckets, each identified by a URI
- Objects are byte containers
- Clients read and update objects / buckets by SOAP / REST-based interface (structure similar to filesystem)

From local DBMS to S3-based DBMS

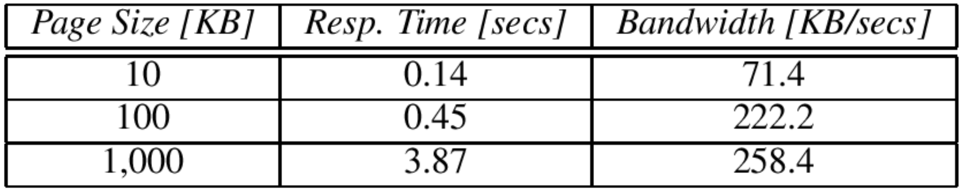
Since interface the same: Just exchange Disk with S3?



But....

Big trouble!!!

- Classical DB Engine updates often and early:
  > Huge transaction costs!
  > Slow (Latency)!



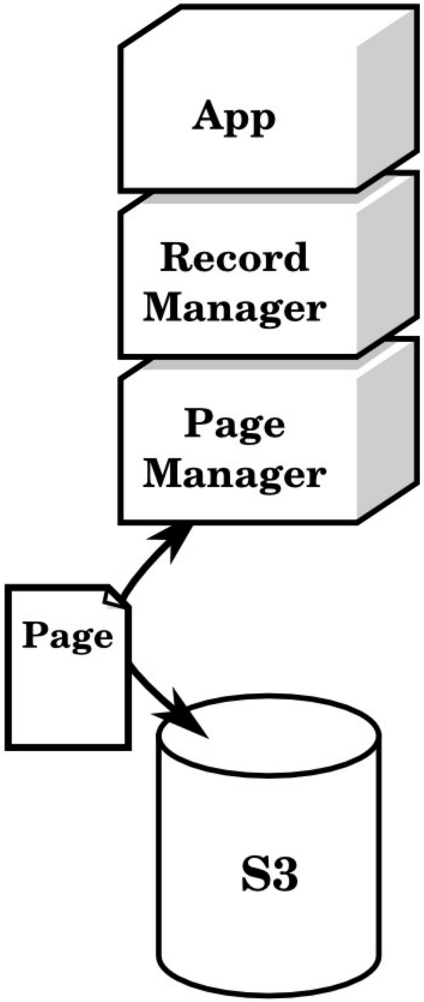
Page Size [KB]	*Resp. Time [secs]*	*Bandwidth [KB/secs]*
10	0.14	71.4
100	0.45	222.2
1,000	3.87	258.4

Table 1: Resp. Time, Bandwidth of S3, Vary Page Size

- S3 only guarantees eventual consistency
  > clients may be overwriting other updates!

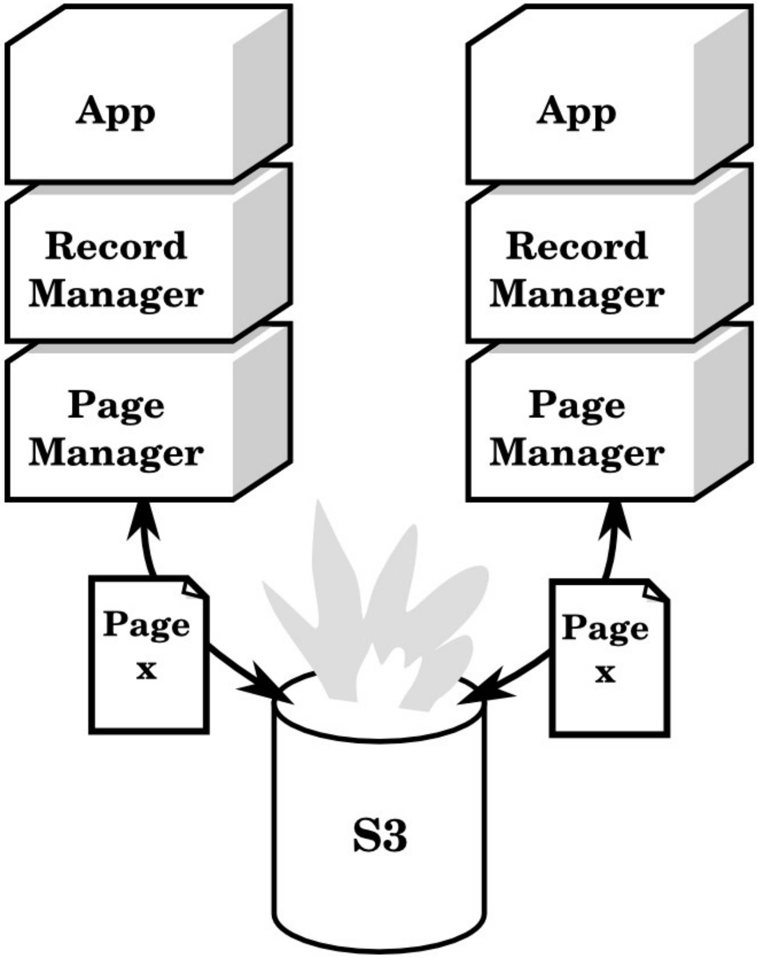
How to reduce transaction cost/latency?

- Add additional 'buffer' layer. In distributed databases this is called paging and – contrary as to what the example may suggest – already widely known:



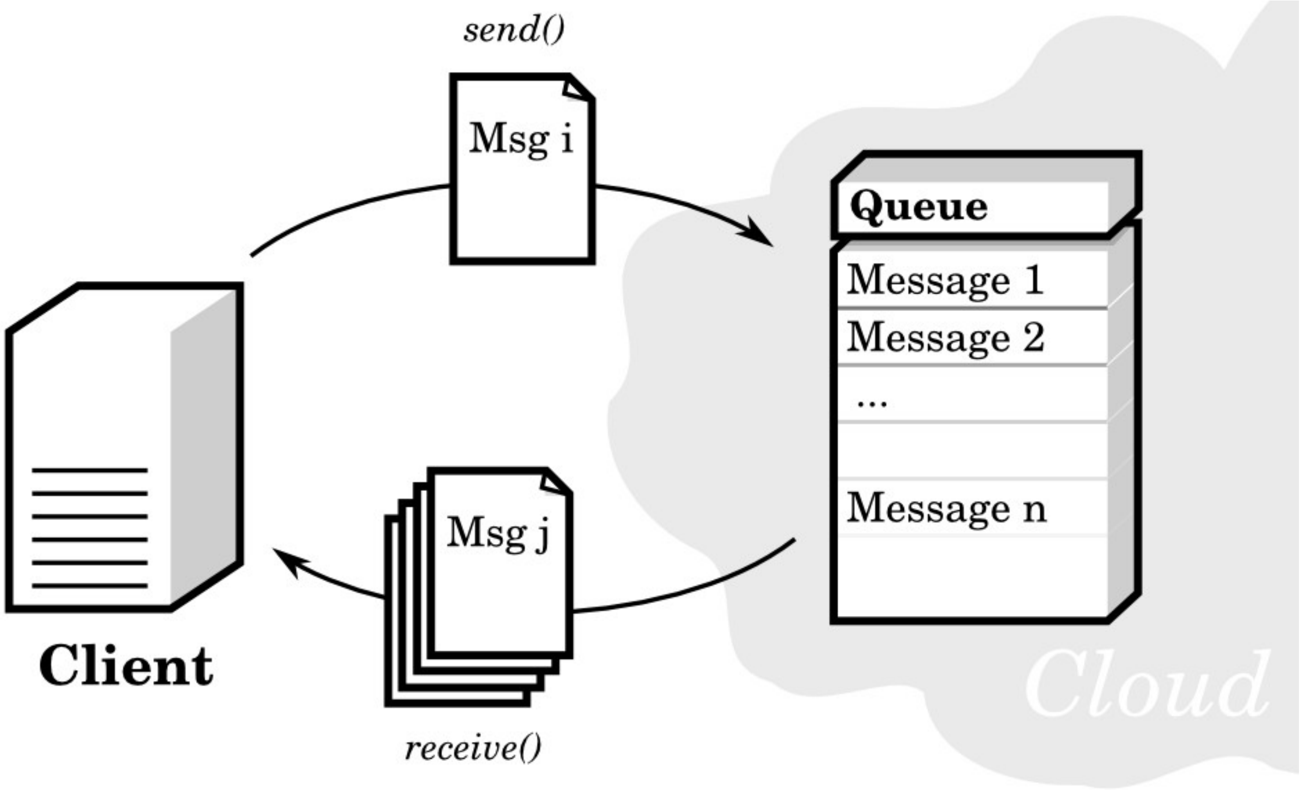
Still some issues:

- Paging is nice but: If two or more clients access pages concurrently, data may be lost.
- If a client waits too long with writing back his buffer, data may be lost if he crashes.



Meet SQS (Simple Query Service)

- Amazons distributed query service
- Availability, reliability and interface similar to S3



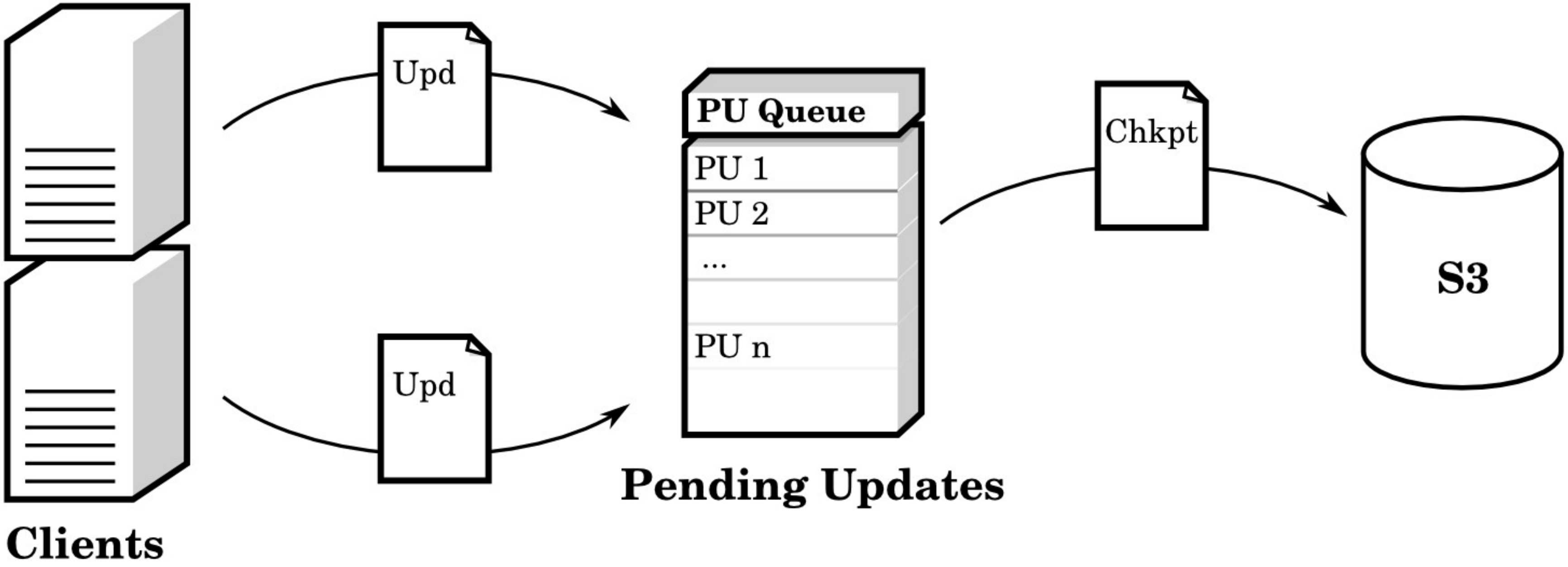
Meet SQS (Simple Query Service)

Functionality:

- Creating unlimited number of queues
- Adding messages (<= 8KB) to queues
- Checking for pending messages ,processed messages are removed
- $0.01 per 10,000 requests (10x cheaper than S3)

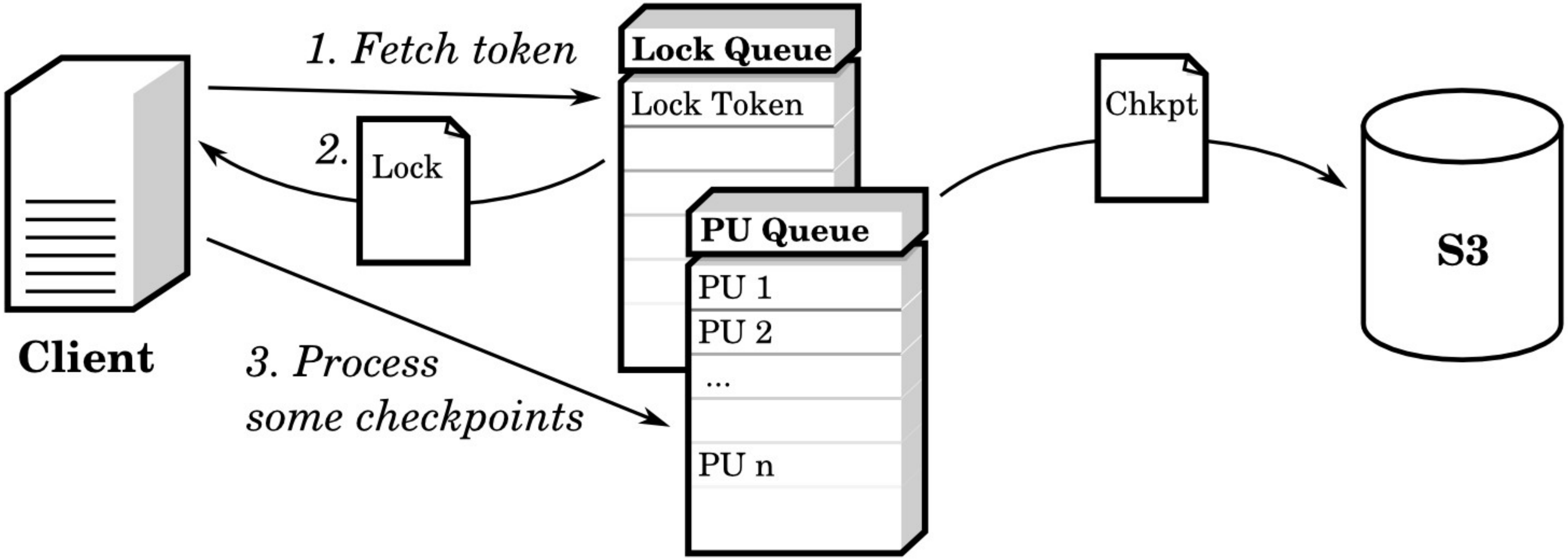
Introducing commit protocols with SQS

- Every page has own PU (Pending updates) queue
- Clients write updates into queue
- PU are written into S3 ('checkpointing') perodically



Commit protocol with SQS cont'd

- Introducing a Lock Queue for every PU Queue to prevent checkpoints being carried out by multiple enitites
- Log records/checkpoints are to be idempotent



How to deal with different data versions?

- Can be affected by requirements
- If you need:
  - Low consistency (monotonic reads):
    Using timestamps on pages
  - High consistency (monotonic writes):
    Using counters on pages -> order checkpoints
  - More freshness:
    Decrease checkpoint interval

- Data never gets lost
- If client crashes while processing checkpoints, updates may be applied twice -> Since updates are idempotent, no data loss happens.

Does such DB on S3 pay?

- Cost per 1000 Transactions: Between 0.15 and 2.9 $ (according to level of consistency)
- Excellent accessibility and scalability
- Unfortunately not (yet) attractive for high-performance transaction processing: too expensive

- Possible solution: Run application on EC2; no transaction costs

Chubby: Googles internal lock service

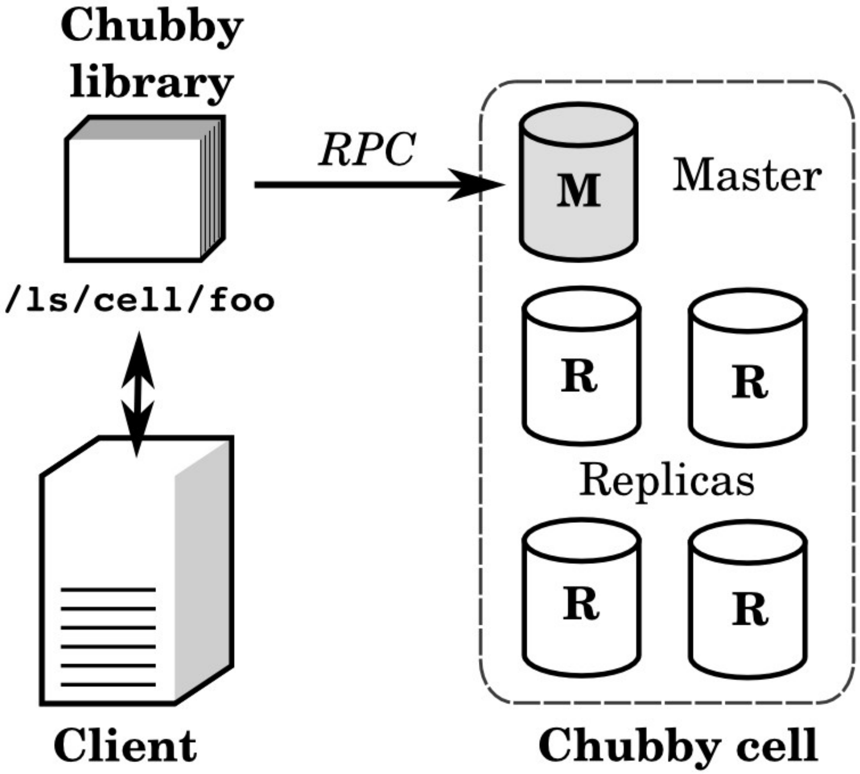
- Purpose: Allow clients to...
  - synchronize activities
  - agree on basic information about environment

- Properties:
  - Reliability, Availability
  - Easy-to-understand semantics
  - Coarse-grained locks (locks for electing a primary, not files)

- Before Chubby was deployed, Google apps used ad-hoc methods or required operator intervention for primary election => Chubby improved situation!

What is Chubbys architecture like?

- Chubby cell of usually 5 nodes
- Master is elected periodically
- Replicas point to Master
- Write requests of clients are propagated to all replicas > Ack if majority of replicas has received it
- Read requests are served by master alone
- Chubby exports a file system with additional services



How a to use Chubby to obtain a lock?

- A new file is generated on the Chubby cell. I.e.:
  ```
  /ls/cell/resource
  ```
- If a client wishes to lock a ressource (or to elect a primary) he connects to Chubby (session) and simply tries to access the file:
  ```
  open('/ls/cell/resource')
  ```
- If the client is successful, he will recieve the file handle.
- The file could now i.e. as well be used to hold the current address of a primary
- (Locking mechanism is advisory!)

How it deals with fail-over?

- Replicas who fail are silently replaced
- If a master fails, remaining replicas re-elect master instantly (usually in seconds) since they poll it frequently
- Clients who hold a session (which has a certain timeout) enter a grace period. If there is a master again before it expires, the continue the session
- Data is restored from replicas
- Memory state (sessions, handles, locks) is conservativly reconstructed with the help of:
  - Stored data on disk
  - States obtained by clients
  - Assumptions

How to deal with different data versions?

- In general: The version with more occurrences wins!
- There are potential faults with the use of locks due to wrong versions:
  - P1 requests Lock L and issues action R1
  - P1 crashes and L is released
  - P2 requests L and issues action R2 on same resource
  - R1 arrives after R2
- Chubby provides a sequencer which holds information about the current lock
- A server providing a locked resource can check if lock is still valid

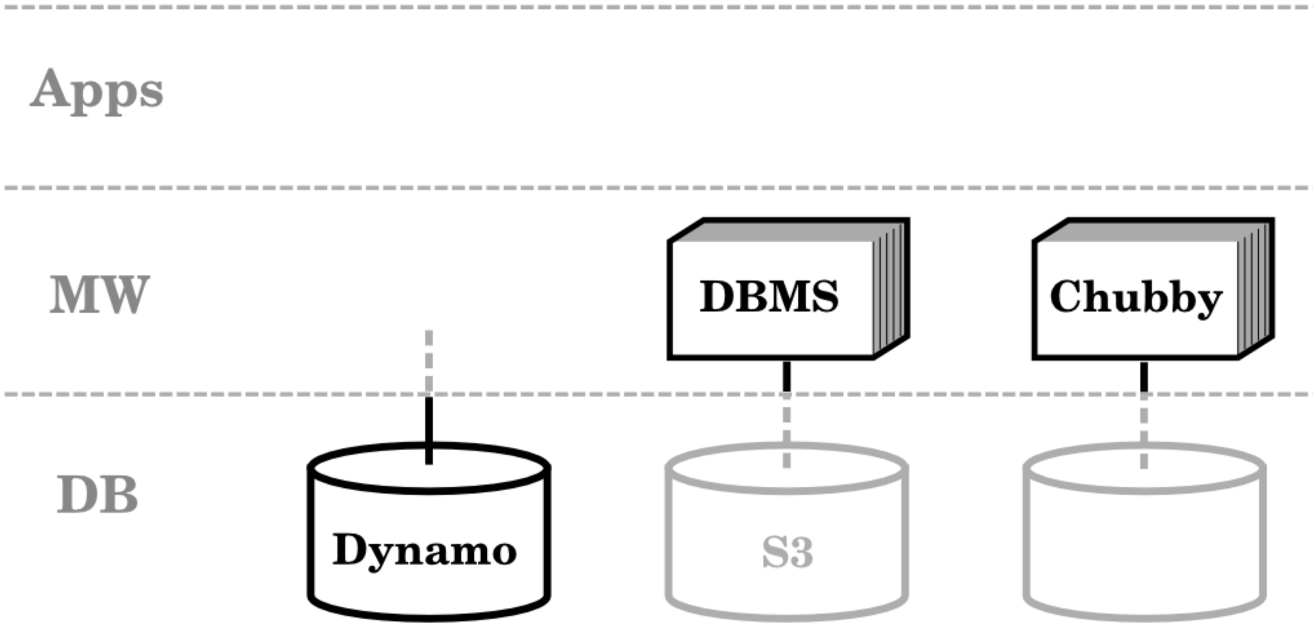
Experiences with Chubby?

- Performs as expected
- Pretty much instantly recovers from failures (most outages were < 15s -> clients didn't even lose session)

- Developers didn't think of abuse, i.e. quota was lacking.
- Most of the clients use it as name server (since it deals well with small TTL) and as repository for configuration files.

The papers in comparison

- Dealing about parts of distributed databases in different levels. More right is more specialized.



Conclusion: Dealing with inconsistencies?

- In distributed databases, accessibility, cost and consistency are diametral.
- System architects have to carefully consider what combination of properties is more important for the applications running on a system
- Dynamo leaves this decision completly to the developer with the parameters N, R, W
- The concept of DB on S3 is basically adaptable to different needs by choosing the level of consistency (# of messages exchanged with queues)
- The Chubby lock service is very specialized and has therefore statical properties concerning the three attributes

References

The presentation was based mostly on these papers:

- **Dynamo: Amazon’s Highly Available Key-value Store**
  Amazon.com, In SOSP 2007.
- **Building a Database on S3**
  Matthias Brantner, Daniela Florescu, David Graf, Donald Kossmann, Tim Kraska, In SIGMOD 2008.
- **The Chubby lock service for loosely-coupled distributed systems**
  Mike Burrows, Google Inc., In USENIX OSDI 2006.

References ...

Hyperlinks to the presented products:

- Amazons S3: http://aws.amazon.com/s3
- Amazons SQS: http://aws.amazon.com/sqs

QUESTIONS
Do you have any?